GSA-SOM: a metaheuristic optimisation algorithm guided by machine learning and application to aerodynamic design

A. González Pérez*; C. B. Allen†; D. J. Poole‡
Department of Aerospace Engineering, University of Bristol, Bristol, UK

Typical algorithms used in optimisation problems can be classified into gradient-based or agent-based optimisers. Gradient-based optimisers boast very fast convergence due to their ability to extract topological information from the local gradient of the objective function, but can often get trapped in a local minimum. Agent-based optimisers use a series of agents to traverse the search space stochastically and are thus much more prone to find the global minimum, but their cost may be unacceptable in typical engineering optimisation problems. The question remains on whether convergence of agent-based algorithms can be accelerated using design space information. A novel optimisation algorithm is introduced that uses machine learning in the form of a Self-Organising Map (SOM) to extract topological information from the optimisation design space and guide the agents of a Gravitational Search Algorithm (GSA). The proposed optimiser is benchmarked against a set of unconstrained analytical optimisation functions, and it is shown to outperform classical agent-based optimisers. The algorithm is further applied to an engineering design problem in the form of constrained transonic aerodynamic shape optimisation, confirming its robustness and higher performance.

I. Introduction

The solution of numerous engineering problems across several disciplines typically involves the execution of computationally-demanding simulations [1–4]. Habitually, solutions to engineering design problems are non-unique, and it is the duty of the designer to find the best possible solution with respect to some meaningful metric. Available computational resources or simulation time limit the number of simulations that can be performed before a final solution is assimilated.

Optimisation is the branch of engineering dealing with the exploration of the space of all possible solutions to a particular design problem (the design space), aiming to find the solution with the best fitness as evaluated by an objective function. Mathematically, Vanderplaats [5, 6] defines an optimisation problem in the following form:

$$\begin{array}{lll} \text{minimise :} & f(\boldsymbol{\alpha}) & \text{(objective function)} \\ \text{subject to :} & \boldsymbol{g}(\boldsymbol{\alpha}) \leq 0 & \text{(inequality constraint)} \\ & \boldsymbol{h}(\boldsymbol{\alpha}) = 0 & \text{(equality constraint)} \\ & \alpha_i^l \leq \alpha_i \leq \alpha_i^u & \text{(side constraints)} \end{array} \tag{1}$$

where $\boldsymbol{\alpha} = \{\alpha_1, ..., \alpha_i, ..., \alpha_D\}$ is the vector of design variables. The objective function represents the metric to rank possible solutions based on their fitness, whereas constraints define the set of possible solutions. Side constraints represent the boundaries of the search space.

It is then evident that when dealing with costly design problems, an efficient exploration of the underlying design space is imperative, since every objective function evaluation requires a call to the simulation model used. Even when the cost of the objective function is minimal, search space volume increases exponentially with problem size and so an exhaustive exploration of the search space remains impractical; the 'curse of dimensionality'. Attempts to reduce cost of optimisation problems come in two streams: 1) using algorithms to efficiently traverse the search space; 2) reducing the dimensionality of the search space.

Typically, two types of search algorithms are used in optimisation; gradient-based and global search algorithms. Gradient-based algorithms compute the local gradient of the objective function at the evaluation points to determine

*Undergraduate Student, AIAA Student Member, alejandro.gonzalez.2016@bristol.ac.uk
†Professor of Computational Aerodynamics, AIAA Senior Member, c.b.allen@bristol.ac.uk
‡Lecturer, AIAA Member, d.j.poole@bristol.ac.uk

the search direction. The simplest of these are generally variations of steepest descent, Newton methods or conjugate gradients, although more advanced algorithms based on Sequential Quadratic Programming (SQP) are also popular [7, 8]. Gradient-based methods boast fast convergence due to their ability to extract local topological information about the search space from the gradient of the objective function. Unfortunately, this guarantees convergence to a local minimum only, which is often not the global minimum for multimodal problems.

Global optimisation algorithms over the last decades have found inspiration in numerous physical [9–11] and natural [12–15] phenomema. They use a number of agents to search the design space, the positions of which get updated throughout iterations based on their relative fitness and some stochastic parameter to ensure convergence is independent of the initial position of the search agents. Global optimisation algorithms thus lie within the field of heuristics and provide greater success at finding the global minimum at the expense of an increase in function calls.

Minimising the cost of global optimisation algorithms has been attempted through surrogate modelling [16–19] by using a limited number of objective function evaluations to generate a statistical model of the design space. The surrogate model can be explored instead of the physical design space, thus significantly reducing the computational cost of the optimisation. Unfortunately, the fidelity of surrogate models is greatly compromised by the dimensionality of the problem, and the number of function evaluations required to generate a surrogate model of high-dimensional spaces is usually greater than that required by the optimiser itself [20].

Reducing the dimensionality of the problem can alleviate these issues to some extent. Considerable efforts have been made over the last decades to develop a robust dimensionality reduction method for optimisation. Principal Component Analysis (PCA) [21], generally through Proper Orthogonal Decomposition (POD), has been successfully demonstrated to reduce the dimensionality of the optimisation problem, both as a procedure to generate reduced order models of the design space [22–24] and, more recently, as an efficient parameterisation method [25, 26].

Unfortunately, for high-dimensional problems, surrogate models lack enough fidelity to justify the savings in computational cost and further efforts are needed to obtain a robust optimisation within the computational budget. The objective of this paper is to introduce a new global optimisation algorithm that uses machine learning to guide the direction of the search. Firstly, the machine learning framework is presented, followed by its integration within the optimisation algorithm. The robustness of this novel approach is then demonstrated against a series of analytical functions. Finally, an application of the algorithm in a set of aerodynamic design problems is presented to demonstrate its performance in a real-world engineering design problem.

II. Self-Organising Maps

Artificial Neural Networks (ANNs) are computational models consisting of a series of processing elements, or neurons, that respond to input data based on some set of predefined activation functions [27]. Each neuron has an associated weight vector, encoding topological information about the input space. ANNs must be trained to match the input space using either unsupervised (using the input data only) or supervised (using input-output data pairs) learning.

Self-Organising Maps (SOMs) [28] are a type of unsupervised ANN consisting of a single layer of competitive neurons that provide a discretised representation of an input space of training data. Training of the SOM follows a simple philosophy: similar data points in the input training space must be associated with nodes that are closer in the grid, whilst more dissimilar data must be situated gradually away in the competitive layer [29].

The central idea surrounding training of the SOM is that every input value must be associated to the node that best matches it, or Best Matching Unit (BMU), and both the node and its neighbours in the computational grid must be updated for better matching. The SOM thus exhibits two characteristics that differentiate it from other ANNs: 1) training happens through competitive learning (neurons must compete to represent subsets of the input data) and 2) the topological ordering of the input space is preserved thanks to the existence of a neighbourhood relationship amongst neurons [30]. A schematic of the SOM is provided in Figure 1.

Since the competitive nature of the training of the SOM results in clustering of data with similar features, a SOM could be used to identify the most promising areas within the design space of an optimisation problem. This is possible since the SOM retains the topological relationships of the design space and thus allows to not only identify the existence of promising regions of the design space in the competitive layer, but also to locate and link back to the corresponding regions in the input design space. SOMs can therefore be regarded as a method for dimensionality reduction; in fact, they could be considered mathematically as a non-linear extension of PCA [31]. The potential of the SOM for identification of topological properties [32–34] and clustering of data [35, 36] has been widely demonstrated in literature, and coupling with optimisation algorithms has proved successful for improving the training strategy for the SOM [37, 38].

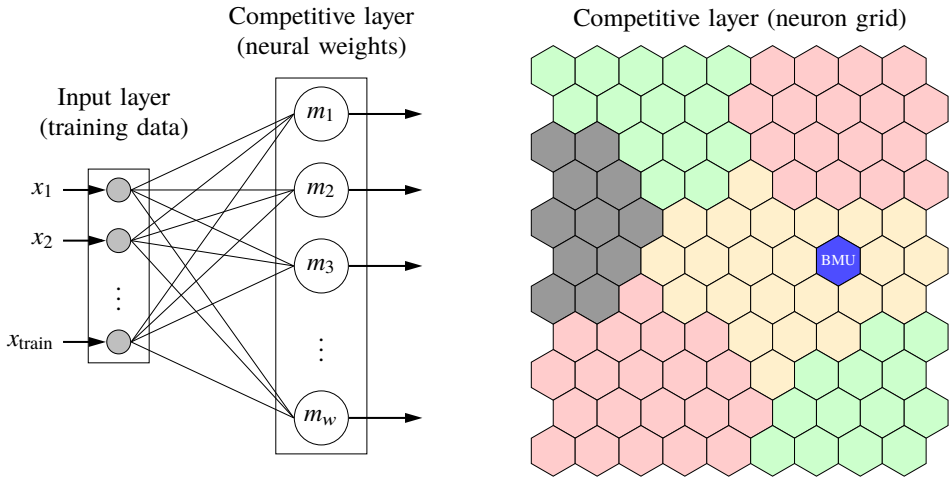


Fig. 1 Schematic of the Self-Organising Map.

A. Stepwise training of the SOM

Training of the SOM was originally presented by Kohonen as a stepwise, recursive procedure [28] where each training point in the input dataset is sent to the SOM once. The BMU is found as the node such that the distance between its associated weight and the input vector is the smallest of all nodes. Let c be the index of the BMU in a competitive layer of n_w nodes. Let $\{\mathbf{m}_i(t)\}$ be a sequence of n-dimensional real vectors representing the nodal weights in the competitive layer at any training step t. Given a real, n-dimensional input vector $\boldsymbol{x}(t)$, the index of its BMU is:

$$c = \underset{i \in [1, n_w]}{\arg\min} \{||\boldsymbol{x}(t) - \boldsymbol{m}_i(t)||\} \tag{2}$$

The weights of the BMU and its neighbouring neurons are then updated to better match the input vector. A neighbouring relationship between neurons must therefore be defined before the updating can take place. The gaussian neighbourhood function – Eq. (3) – has been used in this study for its simplicity and popularity in other SOM implementations. Let $\boldsymbol{r}_i$ be the bidimensional position vector of the ith node in the competitive layer, the gaussian neighbourhood function between the ith and jth nodes is defined as:

$$h_{ij}(t) = \begin{cases} \exp\left(\dfrac{-||\boldsymbol{r}_i - \boldsymbol{r}_j||^2}{2\sigma^2(t)}\right) & \text{if } ||\boldsymbol{r}_i - \boldsymbol{r}_j|| < \sigma(t) \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where $\sigma(t)$ is the influence radius of each node in the net at the current training iteration t. The mathematical nature of the influence radius is not important in itself, provided that it is a monotonically decreasing function. This is imperative since a large initial support radius is necessary to establish the topological order of the map during the initial training phases, whereas a smaller training radius improves the statistical accuracy of the SOM in the later training stages. Eq. (4) shows one such function where the influence radius decreases from an initial value of σ_0 to a value of unity after n_t iterations.

$$\sigma(t) = \sigma_0 \exp\left(-\ln(\sigma_0)\frac{t}{n_t}\right) \tag{4}$$

Once the neighbourhood relationship has been established, the updating procedure can be outlined. The stepwise computation of the SOM requires the introduction of a learning term $\alpha(t)$, which must decrease monotonically throughout training (similarly to the influence radius). The updating procedure for the nodes in the SOM, after the BMU with index c has been found, is presented in Eq. (5). It must be noted that, since the neighbourhood function is null outside of the influence radius, only the nodes closer to the BMU get updated.

$$\boldsymbol{m}_i(t+1) = \boldsymbol{m}_i(t) + \alpha(t) h_{ci}(t) \left[\boldsymbol{x}(t) - \boldsymbol{m}_i(t)\right] \tag{5}$$

The expert reader may be familiar with Eq. (5) from an optimisation background. This expression is in fact similar in nature to the updating procedure of gradient descent optimisation. A direct corollary of this parallelism is that the dimensionality reduction encoded in the competitive layer of the SOM is an optimal representation of the topology of the input layer. Since gradient-based is a local optimisation algorithm, it is also expected that the resulting SOM be dependent on the order of data input and the initial conditions of the SOM.

Mu-Chun Su, Yu-Xiang Zhao, and Lee [39] pioneered the usage of SOMs in optimisation by presenting an algorithm exploiting the analogous behaviour between gradient descent and training of the SOM. Said algorithm replaced the BMU identification procedure with stochastic identification of the neuron with the best fitness. This resulted in an improved gradient-descent algorithm that makes use of the SOM dimensionality reduction to avoid convergence to the nearest local minimum. Unfortunately, this modified version of the SOM loses the ordering potential of the original SOM, and the performance of the algorithm is severely compromised by increasing dimensionality.

B. Batch training of the Self-Organising Map

By considering the convergence of the stepwise training process to a stable state of the SOM, a mathematical expression can be derived for the asymptotic value of the nodal weights: Eq 6.

$$\boldsymbol{m}_i \rightarrow \frac{\sum_{t=1}^{n_t} h_{ci}(t)\boldsymbol{x}(t)}{\sum_{t=1}^{n_t} h_{ci}(t)} \tag{6}$$

It must be noted that this expression is an implicit relationship, since the index c of each winning node is still dependent on an individual $\boldsymbol{x}(t)$ and $\boldsymbol{m}_i(t)$. The sum of n_j input parameters with a common BMU of index j can be approximated by a weighted mean $\bar{\boldsymbol{x}}_j$. Mathematically:

$$\boldsymbol{m}_i(t+1) = \frac{\sum_{j=1}^{n_w} n_j h_{ij}(t)\bar{\boldsymbol{x}}_j(t)}{\sum_{j=1}^{n_w} n_j h_{ij}(t)} \tag{7}$$

This expression is the explicit equivalent to Eq 6. It can be easily proved that both expressions converge to the same steady-state result, but Eq. 7 does so faster since the whole training set is traversed at once, thus requiring just a few or a few dozens of iterations to converge. This updating procedure is also safer, since no learning rate comes into play and the weights are updated by weighted averages of the input data set instead, hence making the resulting SOM independent of the order of traversal of the input data or the initial weights of the SOM. This training procedure is known as the batch computation of the SOM, and has been used in the implementation of the SOM described in this report.

The implementation of the batch training of the SOM simply consists of a list associated to each node. Each training point is added to the list associated to its BMU. Once the whole set of training has been traversed, the nodal weights get updated to a weighted average of the training data in the lists of the nodes lying within its influence radius, as shown in Eq. 7. This process is then repeated a number of times, reducing the influence radius after every iteration.

A significant corollary of the batch training principle is that evaluations of an input vector's BMU are completely independent of each other during training. This results in an algorithm that lends itself particularly well to parallelisation. In fact, due to the simplicity of the training algorithm and since training data need only be sent once to the training algorithm to update the nodal weights of the whole map, the training algorithm can be implemented for execution in GPU with at most two communications between CPU and GPU. The performance improvements of such implementations have been widely demonstrated in literature [40, 41].

III. The GSA-SOM algorithm

A. Gravitational Search Algorithm

The Gravitational Search Algorithm (GSA) is an agent-based global search algorithm mimicking the physical laws of Newtonian gravity to converge to an optimal result [11]. In Rashedi, Nezamabadi-pour, and Saryazdi's algorithm, each particle is assigned a mass based on its fitness, and the relative attraction among all the masses is used to update the positions of the agents.

The simplicity of the algorithm, alongside its robustness and superior performance, makes it a suitable candidate to explore the potential of SOMs in global optimisation. A SOM-guided modification of the classical GSA algorithm is presented here: the GSA-SOM algorithm. Physically, the GSA-SOM algorithm can be regarded as representing the gravitational interaction of masses (GSA) submerged in a gravitational potential field (SOM).

Given a set of n_g agents initialised randomly within the boundaries of the design space, the mass of each agent as per the GSA algorithm is given by the fitness of the agent:

$$m_i(t) = \frac{\mathrm{fit}_i(t) - \mathrm{worst}(t)}{\mathrm{best}(t) - \mathrm{worst}(t)} \tag{8}$$

where the best and worst fitnesses are taken from the fitnesses of the agents at the current iteration. The masses are then normalised based on their relative values:

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{n_g} m_j(t)} \tag{9}$$

The accelerations of the particles are then calculated using Newton's law of universal gravitation. It must be noted that newtonian gravitational attraction decays with the square of the distance between the masses, and that this influence decay is not desirable for the transmission of information between the optimisation agents, so a modified version of newtonian gravity that decays linearly in distance is used. The force $\boldsymbol{F}_{ij}(t)$ exerted on the ith agent by the jth agent, located at positions $\boldsymbol{x}_i(t)$ and $\boldsymbol{x}_j(t)$ respectively, at the evolution t of the algorithm is given by:

$$\boldsymbol{F}_{ij}(t) = G(t)\frac{M_i(t)M_j(t)}{\left\|\boldsymbol{x}_i(t) - \boldsymbol{x}_j(t)\right\|}\left(\boldsymbol{x}_j(t) - \boldsymbol{x}_i(t)\right) \tag{10}$$

where $G(t)$ corresponds to the "universal gravitational constant", which despite its nomenclature decreases over evolutions of the algorithm to permit convergence. Let G_0 be the initial value of the gravitational constant and α, the gravitational decay parameter. The value of $G(t)$ for an evolution t of a total of n_t algorithm evolutions can be expressed as:

$$G(t) = G_0 \exp\left(-\alpha \frac{t}{n_t}\right) \tag{11}$$

The total force acting on each particle is then a stochastic summation of all the individual forces. To prevent less feasible particles from influencing more feasible particles excessively, a discriminating procedure was suggested by Rashedi, Nezamabadi-pour, and Saryazdi whereby the number of particles exerting a force on the others decreases over time. Thus, only the forces caused by the $K_{\mathrm{GSA}}^{\mathrm{best}}$ agents with the best feasibilities act upon the agents, where $K_{\mathrm{GSA}}^{\mathrm{best}}$ is a scalar that linearly decreases from n_g to 2% of the initial value. Thus, let $\mathbf{rand}_j$ be an n-dimensional vector of random variables with uniform probability in the range $[0, 1]$, the force acting on particle i is:

$$\boldsymbol{F}_i(t) = \sum_{j=1}^{K_{\mathrm{GSA}}^{\mathrm{best}}} \mathbf{rand}_j \circ \boldsymbol{F}_{ij}(t) \tag{12}$$

where $\circ$ defines the element-wise multiplication operator. The updating procedure of the algorithm is based in Newtonian mechanics, with particle accelerations $\boldsymbol{a}_i^{\mathrm{GSA}}(t)$ defined as:

$$\boldsymbol{a}_i^{\mathrm{GSA}}(t) = \frac{F_i(t)}{M_i(t)} \tag{13}$$

This value is used to update the particle velocity $\boldsymbol{v}_i(t)$ in a stochastic fashion:

$$\boldsymbol{v}_i(t+1) = \mathbf{rand}_i \circ \boldsymbol{v}_i(t) + \boldsymbol{a}_i^{\mathrm{GSA}}(t) \tag{14}$$

The position of each search agent is then updated accordingly:

$$\boldsymbol{x}_i(t+1) = \boldsymbol{x}_i(t) + \boldsymbol{v}_i(t+1) \tag{15}$$

Particles that fall outside of the search space are reinitialised randomly within the search bounds. The fitness of each particle is then re-evaluated using their new positions and the updating process repeated until a certain number of iterations are completed or convergence has been achieved based on some criterion (*e.g.* standard deviation of the most recent solutions).

B. Self-Organising Map-guided Gravitational Search Algorithm (GSA-SOM)

The GSA algorithm provides a simple yet robust global optimisation algorithm with superior efficiency compared to other algorithms commonly used in optimisation; Rashedi, Nezamabadi-pour, and Saryazdi demonstrated the superior performance of the algorithm against popular Particle Swarm Optimisation (PSO) for the optimisation of a suite of analytical functions [11] and an engineering problem [42].

A significant property of the GSA algorithm is that it is a memory-less algorithm, where the updating procedure of the search agents is based solely on the current position of the particles, resulting in an effectively “blind" exploration of the search space. Sun and Peng [43] proposed a PSO-GSA hybrid that makes use of the collective memory property of the PSO algorithm, whereby search agents remember their historical best position and use it to update their positions. These modifications of the original GSA algorithm resulted in an overall improvement in robustness; however, the best historical position of the agents does not provide enough information about the search space to overcome the “blindness" of the agents.

In this section, a novel optimisation framework that uses a Self-Organising Map for extraction of knowledge from the search space is presented. This knowledge is used to guide the search agents in the GSA algorithm, providing an intelligent exploration of the design space.

The Self-Organising Map-guided Gravitational Search Algorithm (GSA-SOM) uses the historical positions of the GSA search agents to train a SOM on each iteration. The nodal weights are then treated as a separate set of masses and used to update the position of the search agents.

A set of training vectors is needed to train the SOM at each iteration. These are taken from the historical positions of the search agents, up to a total of n_{train} vectors. Each training point is an $(n+1)$-dimensional vector resulting from concatenating the n-dimensional position vector of the corresponding agent and its fitness. Since only the regions of the search space with best fitnesses are relevant, the SOM is trained using the agents' historical best positions. Thus, the SOM can extract the topological relationships amongst them and map the most promising areas, guiding the agents towards them and hence reducing the dimensionality of the search space.

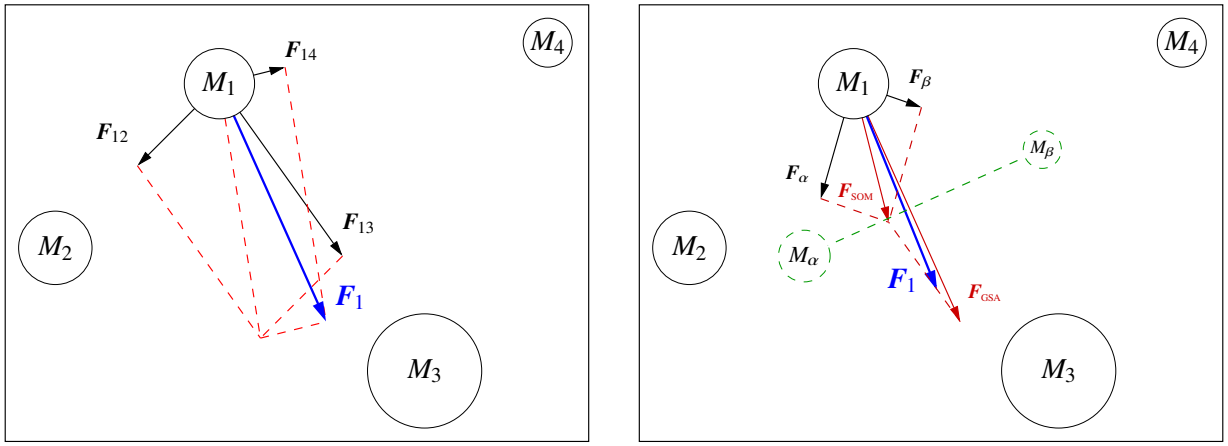
The iterative training of the SOM allows the competitive layer to quickly map the gross features of the search space during the initial iterations, where not only the velocities and accelerations of the search agents are larger, but also the influence radius of the neurons in the competitive layer. As only the best n_{train} historical positions are used in training, the influence of less promising areas on the SOM eventually decays, but their contribution to the global ordering remains.

The map is trained once at each iteration using the batch training algorithm (Eq. 7). The resulting nodal weights are then extracted and considered as an alternative set of n_w masses; these are $(n+1)$-dimensional vectors, the first n dimensions of which correspond to the position of the masses, and the remaining dimension, to their fitness. The masses of the SOM nodes are calculated through the classical GSA algorithm independently of the search agent masses using Eqs. 8 and 9, and the forces they exert on the search agents using Eqs. 10-12. An elitist check is applied in the same manner as in classical GSA such that only the $K_{\mathrm{best}}^{\mathrm{SOM}}$ masses with the highest fitness produce acceleration on the search agents. In the GSA-SOM algorithm, the acceleration of each search agent is therefore a weighted sum between the accelerations induced by the other search agents and the accelerations induced by the SOM nodal weights:

$$\boldsymbol{a}_i(t) = (1-w)\,\boldsymbol{a}_i^{\mathrm{GSA}}(t) + w\boldsymbol{a}_i^{\mathrm{SOM}}(t) \tag{16}$$

Physically, the SOM can be thought of as a gravitational potential field representing the topology of the search space. The masses from the SOM do not interact with one another directly, but rather redirect the agents towards the most feasible regions of the search space. Since the labour of the SOM is to guide the search agents rather than search

the design space itself, it is important that the weighting w of the accelerations caused by the SOM nodes be smaller than the weighting $(1 - w)$ of the accelerations caused by the search agents themselves. Figure 2 shows a comparison between the update framework for the GSA and GSA-SOM algorithms.



(a) GSA (b) GSA-SOM

Fig. 2 Update frameworks of the GSA and GSA-SOM algorithms

The pseudocode for the GSA-SOM algorithm is shown in Algorithm 1 for n_g search agents and n_w competitive neurons. The algorithm is executed for a total number of n_t iterations or until some convergence criterion is met. Figure 3 shows the flow chart of the GSA-SOM algorithm.

Algorithm 1: GSA-SOM Algorithm

```
Initialise n_i particles at random positions within search space boundaries
Initialise Self-Organising Map of n_w neurons with random weights
for t = 1 → n_t do
    for i = 1 → n_g do
        Evaluate fitness of particle i from objective function
        Evaluate mass of particle i: Eqs. 8 and 9
        Calculate GSA acceleration of particles: Eqs. 10-12
        if size(training set) < n_train or fit(i) better than worst(training set) then
            Insert location and fitness of particle i in training set
        end
    end
    Crop training set to n_train vectors
    Train Self-Organising Map: Eq. 7
    Identify best and worst fitness of the Self-Organising Map
    for i = 1 → n_w do
        Evaluate mass of SOM weight i: Eqs. 8 and 9
        Calculate SOM acceleration of particles: Eqs. 10-12
    end
    for i = 1 → n_g do
        Calculate acceleration of particle i: Eq. 16
        Update position and velocity of particle i: Eqs. 14 and 15
    end
end
```

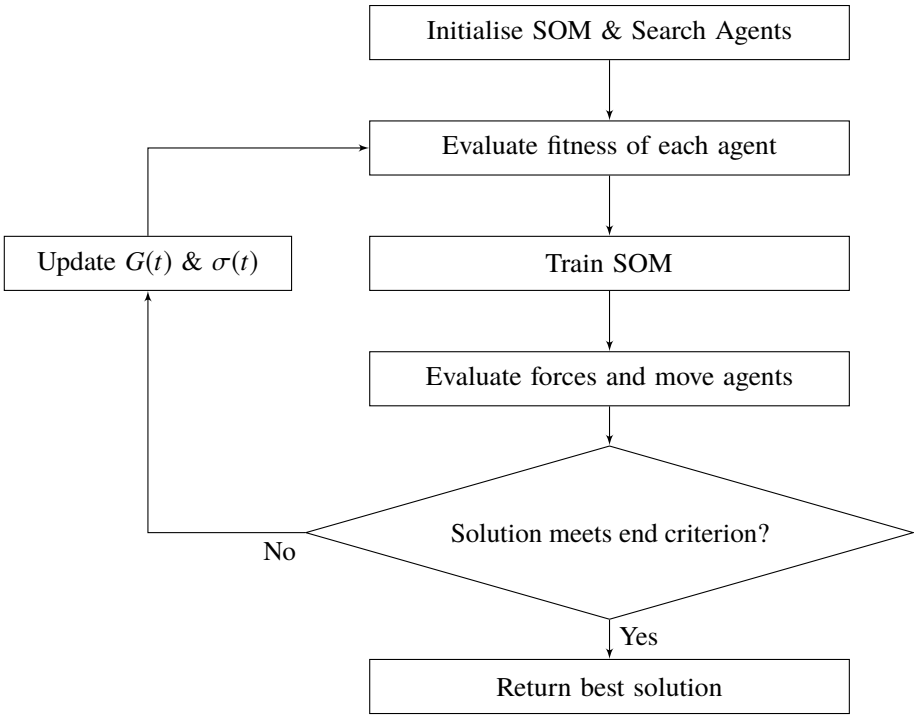


Fig. 3 The GSA-SOM algorithm.

C. Complexity of the algorithm

The GSA-SOM algorithm is implemented as a base GSA optimiser coupled to a SOM. In order to understand the computational time complexity of the algorithm, both modular components must be analysed separately, and their interaction carefully considered.

Training of the SOM takes three steps: a) computing the distances between input vectors and nodal weights, b) updating the training list of the BMUs and c) updating the nodal weights. The three steps ares linear in number of training points n_{train}, dimensions n and nodes n_w.

Most of the GSA operations are iterations over the population of search agents, all of them linear in dimensions and agent population n_g. There are two exceptions to this; the first arises from the inclusion of an elitist check, which requires sorting agents by fitness, with complexity $O(n_g \log n_g)$. The other exception is the computation of particle forces, which requires each agent to sum the forces caused by a factor of the other agents, resulting in $O(n_g^2)$ complexity.

The GSA-SOM algorithm requires maintaining a set of training vectors consisting of the historical positions and fitnesses of the best particles. With particles are previously sorted by fitness, this is done in linear time $O(\max(n_{\text{train}}, n_g))$. Forces on the search agents by the SOM particles are computed in $O(n_g n_w)$; since the number of neurons is expected to be smaller than the number of agents, the GSA-SOM algorithm is bounded by $O(n_g^2)$, same as classical GSA.

Memory requirements increase by the addition of the SOM to the algorithm and maintenance of the training library, but these costs have linear complexities: $O(n_w)$ and $O(n_{\text{train}})$.

IV. Optimisation of analytical functions using GSA-SOM

The performance of the GSA-SOM algorithm for unconstrained optimisation has been tested against a series of test functions common to many optimisation problems. These are presented mathematically in Table 1 with search space bounds S and number of dimensions n. These functions present a wide range of design space topologies, with varying levels of modality and expected optimiser performance. All functions are shifted in $f(\boldsymbol{x})$ so their minimum value is 0.

Three optimisers were used to minimise the proposed functions: GSA, PSO and GSA-SOM. GSA-SOM was tested using both a sparsely trained configuration with 20 training vectors in the training set and a more densely trained configuration with 2000 training points. Both GSA and GSA-SOM used an initial gravitational constant $G_0 = 100$ with

gravitational decay $\alpha = 20$. Weighting of the SOM forces was $w = 0.3$. The SOM consisted of an 8×8 hexagonal neuron lattice with initial radius $\sigma_0 = 3$. All optimisers used 50 search agents over 1000 evolutions.

Table 1 List of analytical test functions.

Function	S	n	Minimum
Unimodal functions			
$f_1(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^2$	$[-5.12, 5.12]^n$	30	$f_1(\boldsymbol{x}) = 0$ $x_i = 0$
$f_2(\boldsymbol{x}) = (x_1 - 1)^2 + \sum_{i=2}^{n} i(2x_i^2 - x_{i-1})^2$	$[-10, 10]^n$	30	$f_2(\boldsymbol{x}) = 0$ $x_i = 2^{-\frac{2^i - 2}{2^i}}$
$f_3(\boldsymbol{x}) = \sum_{i=1}^{n/4} [(x_{4i-3} + 10x_{4i-2})^2 + 5(x_{4i-1} - x_{4i})^2 + (x_{4i-2} - 2x_{4i-1})^4 + 10(x_{4i-3} + 10x_{4i})^4]$	$[-10, 10]^n$	30	$f_3(\boldsymbol{x}) = 0$ $x_i = 0$
$f_4(\boldsymbol{x}) = \sum_{i=1}^{n} (x_i - 1)^2 + \sum_{i=2}^{n} x_i x_{i-1} - n\frac{(n+4)(n-1)}{6}$	$[-n^2, n^2]^n$	30	$f_4(\boldsymbol{x}) = 0$ $x_i = i(n + 1 - i)$
$f_5(\boldsymbol{x}) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	$[-2.038, 2.038]^n$	30	$f_5(\boldsymbol{x}) = 0$ $x_i = 1$
$f_6(\boldsymbol{x}) = \sum_{i=1}^{n} \sum_{j=1}^{i} x_j^2 + \text{rand}(0, 1)$	$[-65.536, 65.536]^n$	30	$f_6(\boldsymbol{x}) = 0$ $x_i = 0$
$f_7(\boldsymbol{x}) = \sum_{i=1}^{n} \text{rand}(0, 1)x_i + \prod_{i=1}^{n} \text{rand}(0, 1)x_i$	$[-10, 10]^n$	30	$f_7(\boldsymbol{x}) = 0$ $x_i = 0$
Multimodal functions			
$f_8(\boldsymbol{x}) = 418.9829n - \sum_{i=1}^{n} x_i \sin\sqrt{\lvert x_i \rvert}$	$[-500, 500]^n$	30	$f_8(\boldsymbol{x}) = 0$ $x_i = 420.9687$
$f_9(\boldsymbol{x}) = 10n + \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i)]$	$[-5.12, 5.12]^n$	30	$f_9(\boldsymbol{x}) = 0$ $x_i = 0$
$f_{10}(\boldsymbol{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi . x_i)\right) + 20 + e$	$[-600, 600]^n$	30	$f_{10}(\boldsymbol{x}) = 0$ $x_i = 0$
$f_{11}(\boldsymbol{x}) = 10n + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-32.768, 32.768]^n$	30	$f_{11}(\boldsymbol{x}) = 0$ $x_i = 0$
$f_{12}(\boldsymbol{x}) = -\sum_{i=1}^{n} \sin(x_i) \sin^{20}\left(\frac{ix_i^2}{\pi}\right) + 29.630883850324413$	$[0, \pi]^n$	30	$f_{12}(\boldsymbol{x}) = 0$ $x_i = \pi\sqrt{2\frac{2i-1}{2n}}$
$f_{13}(\boldsymbol{x}) = \sin^2(\pi w_i) + \sum_{i=1}^{n-1} (w_i - 1)^2[1 + 10\sin^2(\pi w_i + 1)] + (w_n - 1)^2[1 + \sin^2(2\pi w_n)]$, where $w_i = 1 + \frac{x_i - 1}{4}$	$[0, \pi]^n$	30	$f_{13}(\boldsymbol{x}) = 0$ $x_i = 1$
$f_{14}(\boldsymbol{x}) = 39.16599n + \frac{1}{2}\sum_{i=1}^{n} (x_i^4 - 16x_i^2 + 5x_i)$	$[-5, 5]^n$	30	$f_{14}(\boldsymbol{x}) = 0$ $x_i = -2.903534$

The updating procedure for the PSO algorithm is shown below:

$$\boldsymbol{v}_i(t+1) = K(t)\boldsymbol{v}_i(t) + c_1\mathbf{rand}_{i1} \circ \left(\boldsymbol{p}_i(t) - \boldsymbol{x}_i(t)\right) + c_2\mathbf{rand}_{i2} \circ \left(\boldsymbol{s}(t) - \boldsymbol{x}_i(t)\right) \tag{17}$$

$$\boldsymbol{x}_i(t+1) = \boldsymbol{x}_i(t) + \boldsymbol{v}_i(t+1) \tag{18}$$

where c_1 and c_2 are two constants known as the cognitive and social parameters; $K(t)$ is an inertia weight that decreases with optimiser evolutions, and $\boldsymbol{p}_i(t)$ and $\boldsymbol{s}(t)$ represent the particle's and swarm's historical best positions, respectively. Further details about PSO can be found in [13, 44]. Values of $c_1 = 2$ and $c_2 = 2$ have been taken, with $K(t)$ decreasing linearly from 0.9 to 0.2.

Each optimiser was executed a total of 50 times per function. The results are summarised in Table 2. As can be seen, GSA-SOM outperforms classical GSA for most benchmark functions. When using only 20 training points, GSA-SOM obtained generally better results than for the 2000 training point case, confirming that using a smaller number of training points accentuates the dimensionality reduction properties of the SOM, only mapping the most promising area at each iteration. An exception to this behaviour is appreciated for Rastrigin function ($f_9(\boldsymbol{x})$), with the 2000 training point GSA-SOM outperforming the 20 training point configuration. This behaviour is not repeated for the mathematically similar Griewank function $f_{11}(\boldsymbol{x})$.

Table 2 Minimisation results of benchmark functions.

Function	Minimum	GSA	PSO	GSA-SOM (n_{train} = 20)	GSA-SOM (n_{train} = 2000)	
		8.72E−18	3.56E+00	**2.29E−18**	4.28E−18	Best
	$f_1(\boldsymbol{x}) = 0$	2.21E−17	1.24E+01	**4.01E−18**	7.27E−18	Median
$f_1(\boldsymbol{x})$		3.65E−17	3.13E+01	**6.01E−18**	1.39E−17	Worst
	$x_i = 0$	2.27E−17	1.45E+01	**4.02E−18**	7.71E−18	Mean
		6.45E−18	6.84E+00	**7.93E−19**	2.08E−18	Std. Dev.
		7.66E−17	2.37E+03	**8.94E−18**	3.18E−03	Best
	$f_2(\boldsymbol{x}) = 0$	1.50E−16	1.48E+04	**2.04E−17**	5.07E−01	Median
$f_2(\boldsymbol{x})$		1.18E+00	6.44E+04	**3.24E−17**	2.99E+00	Worst
	$x_i = 2^{-\frac{2^i-2}{2^i}}$	2.36E−02	1.76E+04	**2.05E−17**	7.26E−01	Mean
		1.67E−01	1.27E+04	**6.00E−18**	7.03E−01	Std. Dev.
		3.49E−01	3.70E+02	**2.96E−02**	7.63E−01	Best
	$f_3(\boldsymbol{x}) = 0$	5.98E+00	1.79E+03	**2.96E+00**	4.47E+00	Median
$f_3(\boldsymbol{x})$		1.05E+02	6.17E+03	**1.52E+01**	3.48E+01	Worst
	$x_i = 0$	1.14E+01	2.31E+03	**3.66E+00**	5.92E+00	Mean
		1.82E+01	1.66E+03	**3.20E+00**	6.11E+00	Std. Dev.
		3.14E+04	9.95E+04	**9.70E+03**	5.34E+04	Best
	$f_4(\boldsymbol{x}) = 0$	7.70E+04	2.82E+05	**3.96E+04**	1.07E+05	Median
$f_4(\boldsymbol{x})$		1.57E+05	7.17E+05	**9.68E+04**	2.02E+05	Worst
	$x_i = i(n+1-i)$	8.19E+04	3.10E+05	**4.27E+04**	1.12E+05	Mean
		3.29E+04	1.31E+05	**2.27E+04**	2.85E+04	Std. Dev.
		2.53E+01	1.42E+02	2.56E+01	2.76E+01	Best
	$f_5(\boldsymbol{x}) = 0$	2.61E+01	3.21E+02	**2.59E+01**	2.79E+01	Median
$f_5(\boldsymbol{x})$		2.79E+01	6.28E+02	**2.65E+01**	2.81E+01	Worst
	$x_i = 1$	2.62E+01	3.37E+02	**2.59E+01**	2.79E+01	Mean
		3.56E−01	1.12E+02	1.74E−01	**9.71E−02**	Std. Dev.
		1.00E−01	6.92E+03	**1.54E−02**	2.38E+01	Best
	$f_6(\boldsymbol{x}) = 0$	2.06E−01	2.93E+04	**4.59E−02**	2.83E+02	Median
$f_6(\boldsymbol{x})$		1.29E+00	6.04E+04	**7.69E−01**	1.74E+03	Worst
	$x_i = 0$	2.76E−01	2.92E+04	**7.84E−02**	3.96E+02	Mean
		2.30E−01	1.07E+04	**1.17E−01**	3.16E+02	Std. Dev.
		7.20E−08	6.03E+00	3.89E−09	**2.99E−09**	Best
	$f_7(\boldsymbol{x}) = 0$	2.25E−07	1.03E+01	5.49E−09	**5.26E−09**	Median
$f_7(\boldsymbol{x})$		1.27E−06	2.02E+01	**7.77E−09**	1.02E−08	Worst
	$x_i = 0$	3.17E−07	1.12E+01	**5.60E−09**	5.96E−09	Mean
		2.62E−07	3.59E+00	**9.52E−10**	1.81E−00	Std. Dev.

Table 2 Minimisation results of benchmark functions (continued).

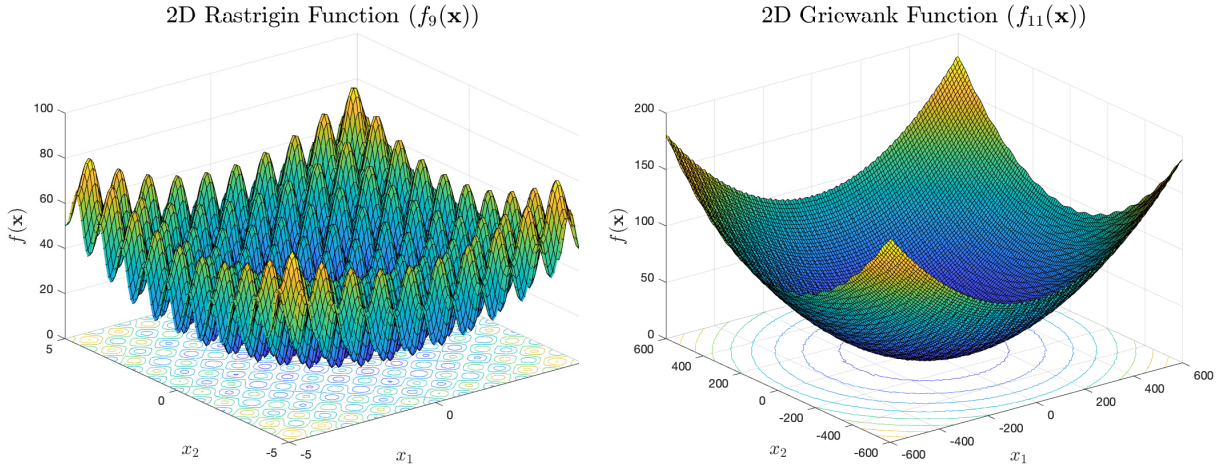
Function	Minimum	GSA	PSO	GSA-SOM (n_{train} = 20)	GSA-SOM (n_{train} = 2000)	
$f_8(\boldsymbol{x})$	$f_8(\boldsymbol{x}) = 0$	8.20E+03	**3.41E+03**	8.80E+03	8.95E+03	Best
		9.71E+03	**5.51E+03**	9.67E+03	9.70E+03	Median
		1.05E+04	**6.62E+03**	1.02E+04	1.05E+04	Worst
	x_i = 420.9687	9.65E+03	**5.47E+03**	9.64E+03	9.69E+03	Mean
		4.57E+02	6.46E+02	**3.50E+02**	3.74E+02	Std. Dev.
$f_9(\boldsymbol{x})$	$f_9(\boldsymbol{x}) = 0$	6.96E+00	7.21E+01	3.98E+00	**7.11E−15**	Best
		1.54E+01	1.08E+02	9.95E+00	**1.60E−14**	Median
		2.89E+01	1.39E+02	1.79E+01	**2.98E+00**	Worst
	x_i = 0	1.53E+01	1.05E+02	9.91E+00	**7.36E−01**	Mean
		4.17E+00	1.79E+01	3.11E+00	**8.72E−01**	Std. Dev.
$f_{10}(\boldsymbol{x})$	$f_{10}(\boldsymbol{x}) = 0$	2.27E−09	1.12E+01	**1.20E−09**	1.28E−09	Best
		3.78E−09	1.39E+01	**1.60E−09**	1.79E−09	Median
		4.86E−09	1.61E+01	**1.97E−09**	2.31E−09	Worst
	x_i = 0	3.71E−09	1.36E+01	**1.61E−09**	1.80E−09	Mean
		6.27E−10	1.28E+00	**1.59E−10**	2.22E−10	Std. Dev.
$f_{11}(\boldsymbol{x})$	$f_{11}(\boldsymbol{x}) = 0$	1.26E+00	1.46E+01	**3.33E−16**	7.97E+00	Best
		3.73E+00	4.76E+01	**1.15E+00**	1.74E+01	Median
		8.96E+00	8.57E+01	**3.70E+00**	3.16E+01	Worst
	x_i = 0	3.99E+00	4.84E+01	**1.18E+00**	1.78E+01	Mean
		1.67E+00	1.87E+01	**7.95E−01**	4.94E+00	Std. Dev.
$f_{12}(\boldsymbol{x})$	$f_{12}(\boldsymbol{x}) = 0$	**6.59E−01**	6.45E+00	6.93E−01	1.10E+00	Best
		2.22E+00	9.97E+00	**2.01E+00**	3.78E+00	Median
		4.48E+00	1.45E+01	**3.92E+00**	9.38E+00	Worst
	$x_i = \pi\sqrt{2\frac{2i-1}{2n}}$	2.13E+00	1.01E+01	**2.06E+00**	3.81E+00	Mean
		7.50E−01	1.50E+00	9.83E−01	1.69E+00	Std. Dev.
$f_{13}(\boldsymbol{x})$	$f_{13}(\boldsymbol{x}) = 0$	5.28E−18	3.16E+00	**5.63E−19**	1.58E−18	Best
		8.55E−18	1.17E+01	**1.59E−18**	3.11E−18	Median
		4.54E−01	2.28E+01	**2.44E−18**	5.28E−18	Worst
	x_i = 0	9.09E−03	1.22E+01	**1.60E−18**	3.23E−18	Mean
		6.43E−02	4.11E+00	**3.86E−19**	8.48E−19	Std. Dev.
$f_{14}(\boldsymbol{x})$	$f_{14}(\boldsymbol{x}) = 0$	2.83E+01	1.48E+02	**−5.27E−03***	1.41E+01	Best
		7.07E+01	2.46E+02	**4.95E+01**	5.65E+01	Median
		1.27E+02	3.53E+02	**1.13E+02**	5.70E+02	Worst
	x_i = 0	7.15E+01	2.40E+02	**5.46E+01**	7.53E+01	Mean
		2.58E+01	4.57E+01	3.09E+01	8.36E+01	Std. Dev.

The disparity in optimiser performance for these two functions lies in the search boundaries for each, plotted in Figure 4. Clearly, the effect of multimodality is much more significant for the Rastrigin function as defined by its boundaries; more than 20 points are necessary for the SOM to map the whole design space and capture the more subtle global trend. This is not necessary for Griewank's function, where the quadratic trend dominates and the collective learning of the SOM throughout evolutions can identify it and guide the agents using only 20 training points.

Some of the unimodal functions optimised consisted of steep regions followed by shallow areas near the minimum, for which small variations in some dimensions can produce significant changes of fitness compared to other dimensions. Trid's function ($f_4(\boldsymbol{x})$) is a good example, with PSO getting minima of the order of 10^5. The SOM can only slightly alleviate these performance issues, as its training procedure makes it susceptible to large variations in the input data.

Figure 5 shows a comparison of the convergence behaviour of the GSA-SOM algorithm with classical GSA and PSO. As can be appreciated, GSA-SOM exhibits similar convergence behaviour to that of classical GSA, but it is much less likely to get trapped in a local minimum. Typically, the GSA-SOM algorithm requires a dozen or a few dozen iterations less than GSA to obtain similar results. This characteristic, along with the lower standard deviation of the GSA-SOM optimisation results shown in Table 2, confirms the robustness of the GSA-SOM algorithm.

* f_{14} was shifted so its minimum would be zero, the negative result is due to numerical inaccuracy of the shift.



(a) Rastrigin function, $x_i \in [-5.12, 5.12]$

(b) Griewank function, $x_i \in [-600, 600]$

Fig. 4 2D design space of functions $f_9(x)$ and $f_{11}(x)$.

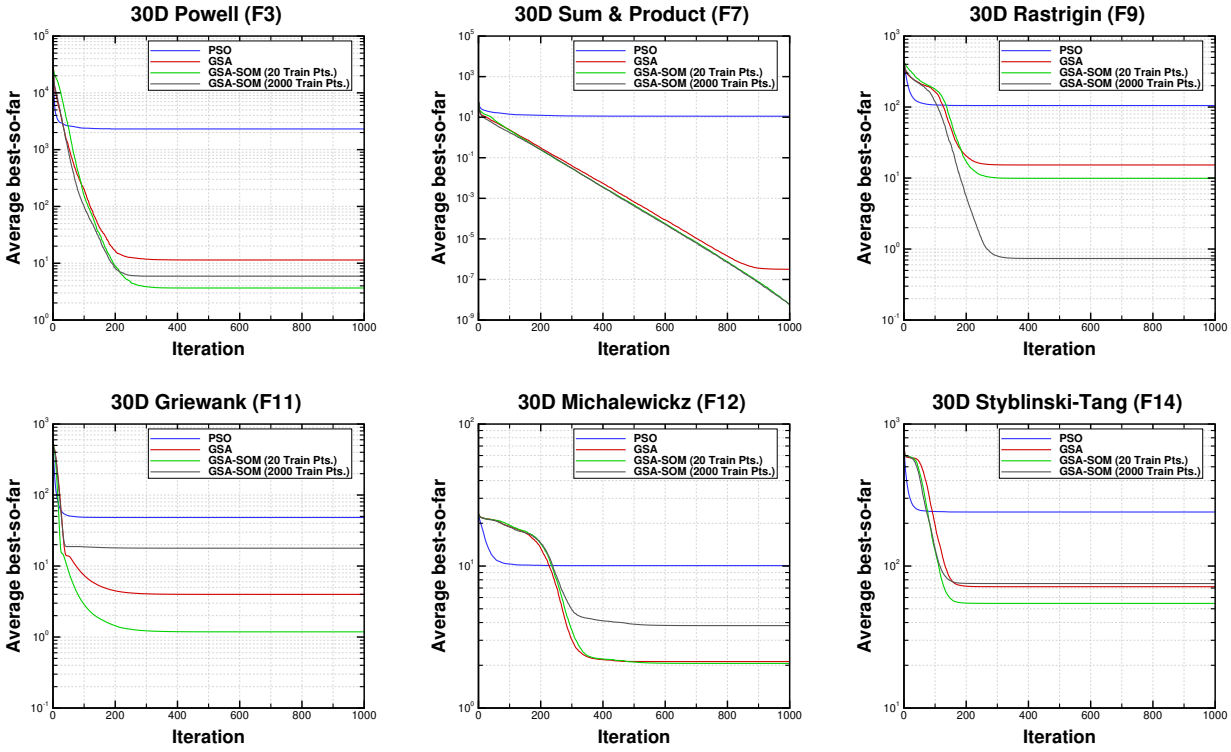


Fig. 5 Convergence plots of minimisation of selected benchmark functions.

V. Aerodynamic Shape Optimisation using GSA-SOM

Having demonstrated the performance of the GSA-SOM algorithm for finding the minimum of a series analytical functions, the next step is to demonstrate the suitability of the algorithm for a typical engineering optimisation problem. A series of benchmark Aerodynamic Shape Optimisation (ASO) cases are presented in this section and the integration of the GSA-SOM algorithm into the optimisation process is discussed.

A. Surface parameterisation and mesh motion

The parameterisation method used in this report was first presented by Poole, Allen, and Rendall [26] and is based on performing a Proper Orthogonal Decomposition (POD) via Singular Value Decomposition (SVD) of a training library of aerofoils. A training library of 1300 aerofoils extracted from the UIUC Airfoil Coordinates Database[†] has been used in this study. All aerofoils are normalised to a chord length of 1 with their leading edge and trailing edge points fixed at $[0,0]$ and $[1,0]$, respectively. The following geometrical distribution was used for all aerofoils [45]:

$$x_i = \frac{1}{4}\left[1 - \cos\left(\pi \frac{i - 150}{150}\right)\right]^2 \quad \forall i \in [0, 300] \tag{19}$$

The UIUC library contained numerous aerofoils of varied quality for the transonic optimisation cases considered. It is expected that the quality of the parameterisation for the optimisation problem be dependent on that of the training library, so a prior filtering of the training library based on performance of the training aerofoils at the transonic regime was performed. This allowed to reduce the dimensionality of the search space (and hence the modality [46]) without impacting the performance of the parameterisation scheme.

The Korn technology factor, κ [47], was used to rank the performance of training aerofoils, such that the 100 aerofoils with highest κ were extracted from the database. SVD was then performed on the training set to obtain the modal deformations. For a library of m aerofoils with a consistent parameterisation, such that every aerofoil has N points with the same distribution of x-coordinates, the matrix of deformations is evaluated as the vector difference of the z-coordinates amongst all aerofoils. The resulting $n \times m_{\text{def}}$ matrix, where $m_{\text{def}} = m(m-1)/2$ corresponds all the possible permutations of aerofoils in the training library, is shown in Eq. 20:

$$\boldsymbol{Z} = \begin{pmatrix} \Delta z_{11} & \cdots & \Delta z_{1 m_{\text{def}}} \\ \vdots & \ddots & \vdots \\ \Delta z_{N1} & \cdots & \Delta z_{N m_{\text{def}}} \end{pmatrix} \tag{20}$$

Given a rectangular matrix Z, its SVD is defined as the following decomposition:

$$Z = U \boldsymbol{\Sigma} V^T \tag{21}$$

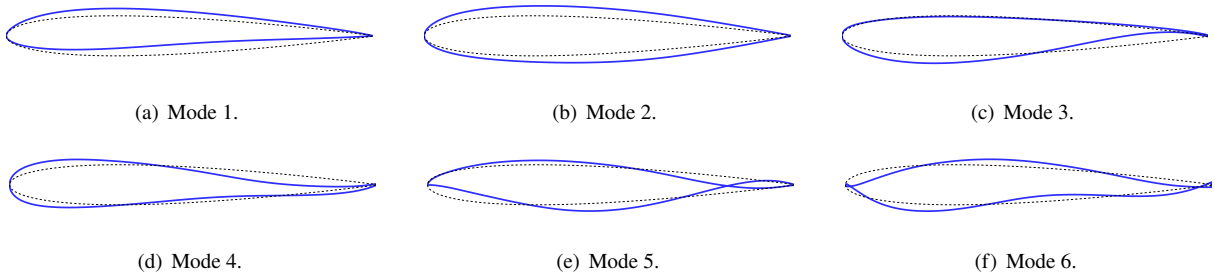
where $\boldsymbol{U}$ is a unitary matrix, the columns of which are modal deformations. $\boldsymbol{\Sigma}$ is a diagonal matrix containing the singular values of X [48] and $\boldsymbol{V}$, a weighting matrix. Physically, the squares of the elements of $\boldsymbol{\Sigma}$ represent the relative energy of the modes in $\boldsymbol{U}$ [21]. A typical criterion for deciding the number of modes to use in the parameterisation is conserving a certain amount of the total energy. In this study, a fixed number of 12 modal deformations was used, providing a satisfactory imbalance between design space coverage and optimiser performance.

New aerofoil shapes are then obtained by adding a weighted sum of the modal deformations (the columns U^i of the left-singular matrix U) to a base aerofoil – as shown in Eq. 22 for n_m modes. The linear weighting coefficients α_i are thus the design variables of the optimisation problem.

$$X \rightarrow X + \sum_{i=1}^{n_m} \alpha_i U^i \tag{22}$$

Figure 6 shows the resulting modal deformations. The first two modes represent parameters similar in nature to thickness and camber. Modes 3 and 4 exhibit characteristics modifying the leading edge roundness and trailing edge thickness distributions (in particular, the third mode depicts the characteristic thickness distribution of supercritical aerofoils). Higher order modes provided different oscillatory deformations necessary to regenerate the training set of aerofoils.

[†] Aerofoil database available online at `https://m-selig.ae.illinois.edu/ads/coord_database.html`.



(a) Mode 1. (b) Mode 2. (c) Mode 3.

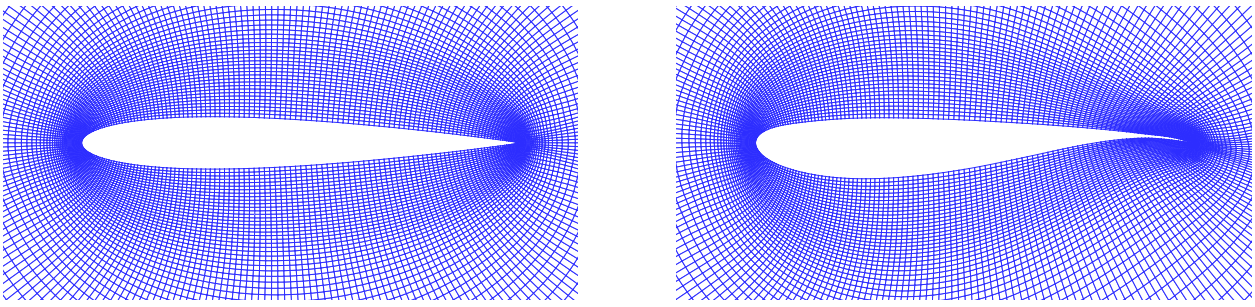
(d) Mode 4. (e) Mode 5. (f) Mode 6.

Fig. 6 Exaggerated modal deformations superimposed to NACA-0012 aerofoil.

A mesh deformation scheme is needed to translate aerofoil deformations to the aerodynamic mesh, and a method based on Radial Basis Functions (RBFs) has been used here. This scheme was first presented by Rendall and Allen [49] and is based on the application of RBFs for interpolation of scattered data presented by Wendland [50] and Buhmann [51].

RBFs translate the deformations of a set of control points to a zone of influence around them. Control points were placed on the aerofoil surface to directly link aerofoil deformations to control point deformations. Masters, Taylor, Rendall, Allen, and Poole [45] showed that 20-25 control points suffice to represent an exhaustive aerofoil library with manufacturing tolerance, so 24 control points are used here.

The interpolation is built using Wendland's C^2 function [50], and a support radius of one chord was used. Figure 7 shows a sample mesh deformation using RBFs.



(a) Undeformed (NACA-0012). (b) Deformed (Mode 3).

Fig. 7 Sample deformation of 257×97 O-mesh for NACA 0012 aerofoil.

B. Flow solver

The flow solver used for the optimisation is a structured multiblock solver solving the compressible Euler (inviscid cases) or Reynolds-Averaged Navier-Stokes (RANS, viscous case) equations. Convective terms are evaluated using a third-order van Leer flux vector splitting [52]. Diffusive terms are evaluated using second order central differencing and turbulence is modelled using the Spalart-Allmaras single equation model [53, 54]. A five-stage Runge-Kutta with local timestepping is used for time integration and convergence is accelerated through V-cycle multigrid [55].

For inviscid cases, single block 257×97 O-meshes were generated, extending to 50 chords at farfield [56, 57]; these are shown in Figure 8. For the viscous optimisation test case, a three-block C-mesh was generated, having 385 points around the aerofoil, 129 points into the farfield and 65 points along the wake line; the C-mesh generated is depicted in Figure 9.

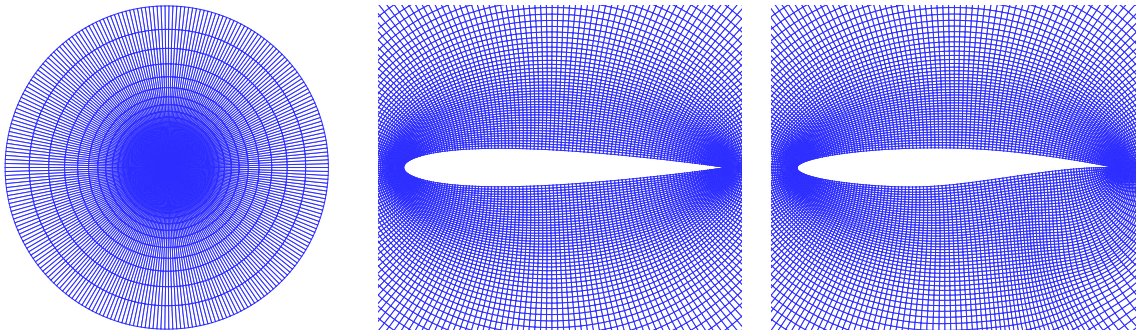


Fig. 8 257×97 **O-mesh for NACA0012 and RAE2822 aerofoils.**

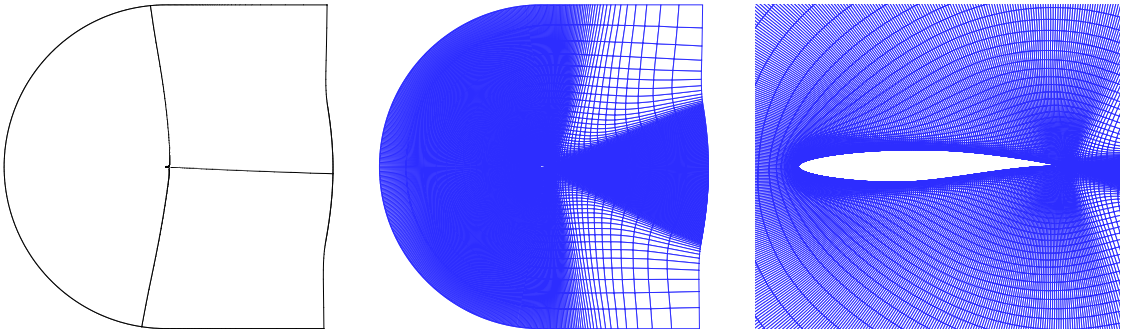


Fig. 9 $385 \times 129 \times 65$ **Three-block C-mesh for RAE2822 aerofoil.**

C. Aerofoil optimisation test cases

A total of five aerofoil optimisation test cases have been studied: four inviscid drag minimisation cases and a viscous drag minimisation case; these are summarised in Table 3. The test cases are based on those presented by Poole, Allen, and Rendall [58], with the viscous optimisation case originally presented by Chernukhin and Zingg [46].

All cases provide examples of constrained optimisation. Constraints are handled using a modified 3S-GSA algorithm [59]. The basis for this method is that feasible and infeasible particles are separated into sub-swarms. Feasible particles are updated using the GSA (or GSA-SOM) procedure, whereas infeasible particles are updated using PSO, with the particle and swarm's best positions decided using a dominance approach: if feasible, the particle with better fitness wins; if unfeasible, the particle with lower constraint violation wins; feasible particles always win unfeasible ones. A new objective function definition is required to account for constraint violations, as shown in Eq. 23. Only feasible particles are used to train the SOM to avoid mapping the infeasible subspace. Full details of the 3S-GSA algorithm can be found in [59, 60].

Table 3 Aerodynamic optimisation test cases.

							Constraints		
	Aerofoil	M_∞	α	C_L	Re	Objective	Lift	Moment	Area
Case 1	NACA 0012	0.700	3.00	0.53	-	C_D	$C_L \geq C_L^0$	$\lvert C_m\rvert \leq \lvert C_m^0\rvert$	$A \geq A^0$
Case 2	RAE 2822	0.700	3.00	1.00	-	C_D	$C_L \geq C_L^0$	$\lvert C_m\rvert \leq \lvert C_m^0\rvert$	$A \geq A^0$
Case 3	NACA 0012	0.850	0.00	0.00	-	C_D	$C_L = 0$	$\lvert C_m\rvert = 0$	$A \geq A^0$
Case 4	NACA 0012	0.650	5.00	1.21	-	C_D	$C_L \geq C_L^0$	$\lvert C_m\rvert \leq \lvert C_m^0\rvert$	$A \geq A^0$
Case 5	RAE 2822	0.729	2.07	0.69	7.0E+06	C_D	$C_L \geq C_L^0$	$\lvert C_m\rvert \leq \lvert C_m^0\rvert$	$A \geq A^0$

$$f^P(\boldsymbol{x}_i) = \begin{cases} f(\boldsymbol{x}_i) & \text{if } \boldsymbol{x}_i \text{ feasible} \\ \sum_{j=1}^{G} \max(0, g_j(\boldsymbol{x}_i)) + \sum_{j=1}^{H} |h_j(\boldsymbol{x}_i))| & \text{otherwise} \end{cases} \tag{23}$$

The classical GSA and GSA-SOM optimisers were used for the drag minimisations using the 3S constraint handling approach. GSA parameters G_0 and α were set to 30 and 10, respectively, and PSO parameters c_1 and c_2, to 2. Poole, Allen, and Rendall [58] proved that these values provided the best results for the 3S-GSA algorithm. A search population of 98 search agents over 1000 evolutions was used for the inviscid case, these values chosen based on the computational budget. Due to the higher cost of the viscous optimisation, only 84 agents over 800 evolutions were used.

The GSA and GSA-SOM optimisers were implemented in parallel. Objective function evaluations of the agents were performed in parallel in the MPI environment. Particle updates and SOM training were performed by the master rank using shared-memory parallelism in OpenMP.

The optimisation results are summarised in Table 4. The GSA-SOM noticeably outperforms the classical GSA algorithm in all test cases. Using 2000 training points provided lower drag for all inviscid cases than using just 20 training points, except for case 2, where the results of both optimisations are fairly similar. This could be indicative of highly multimodal design spaces, where the variations in drag at each of the local minima are more significant than the underlying convexity of the search space. The opposite effect is seen in the viscous case, where 20 training points provide faster convergence, suggesting lower presence of modality in the design space.

Table 4 Aerodynamic drag minimisation results (drag counts).

						Initial	**GSA**		**GSA-SOM (n_{train} = 20)**		**GSA-SOM (n_{train} = 2000)**	
	Aerofoil	M_∞	α	C_L	Re	C_D	C_D	ΔC_D	C_D	ΔC_D	C_D	ΔC_D
Case 1	NACA 0012	0.700	3.00	0.53	-	90.2	10.5	-88.4%	8.9	-90.1%	**7.9**	**-91.2%**
Case 2	RAE 2822	0.700	3.00	1.00	-	68.2	20.8	-69.5%	**19.9**	**-70.8%**	20.1	-70.5%
Case 3	NACA 0012	0.850	0.00	0.00	-	474.2	84.5	-82.2%	62.5	-86.8%	**48.4**	**-89.8%**
Case 4	NACA 0012	0.650	5.00	1.21	-	222.9	95.5	-57.2%	94.2	-57.7%	**88.9**	**-60.1%**
Case 5	RAE 2822	0.729	2.07	0.69	7.0E+06	115.8	106.4	-8.1%	**104.0**	**-10.1%**	106.0	-8.5%

Figure 10 shows the convergence behaviours of the algorithms for the different optimisation cases. The trends evidence the faster convergence of the GSA-SOM algorithm, which can obtain similar results to classical GSA in circa one order of magnitude less objective function evaluations.

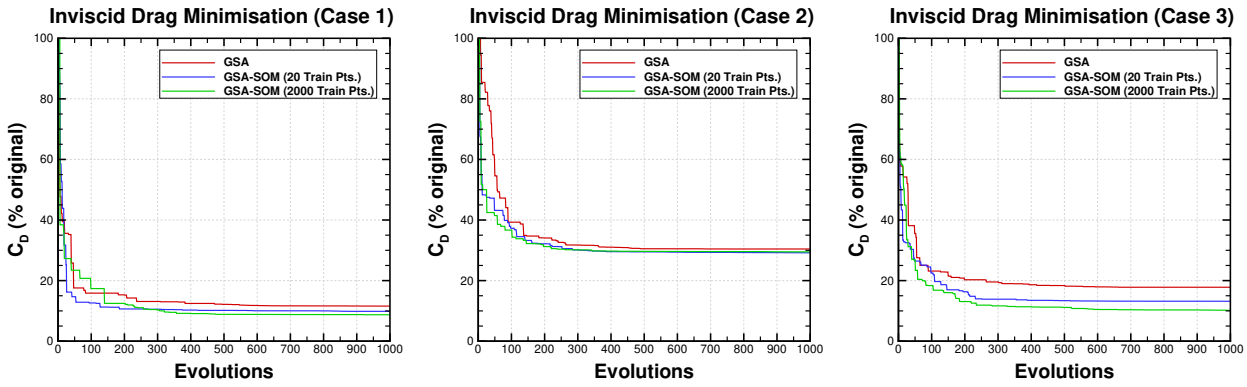


Fig. 10 Convergence plots of inviscid drag minimisation cases.

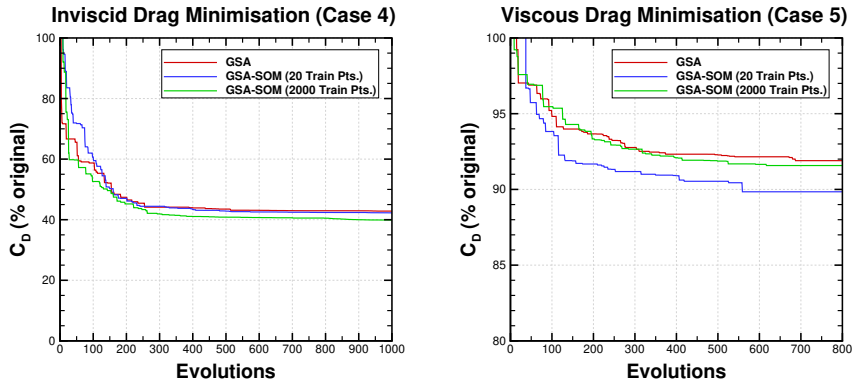


Fig. 10 Convergence plots of inviscid drag minimisation cases (continued).

Optimal aerofoil shapes are presented along their pressure distributions in Figures 11-15. All inviscid cases exhibit some element of oscillatory pressure distributions, a common result in inviscid ASO. Curvature penalties would mitigate this issue. Viscosity provides a natural penalty for oscillatory surfaces, resulting in a smooth surface and pressure distribution as shown in Figure 15. The optimisers successfully eliminate the transonic flow in cases 1, 2 and 5, but are unable to do it for the high free-stream Mach in optimisation case 3 and highly loaded aerofoil in case 4.

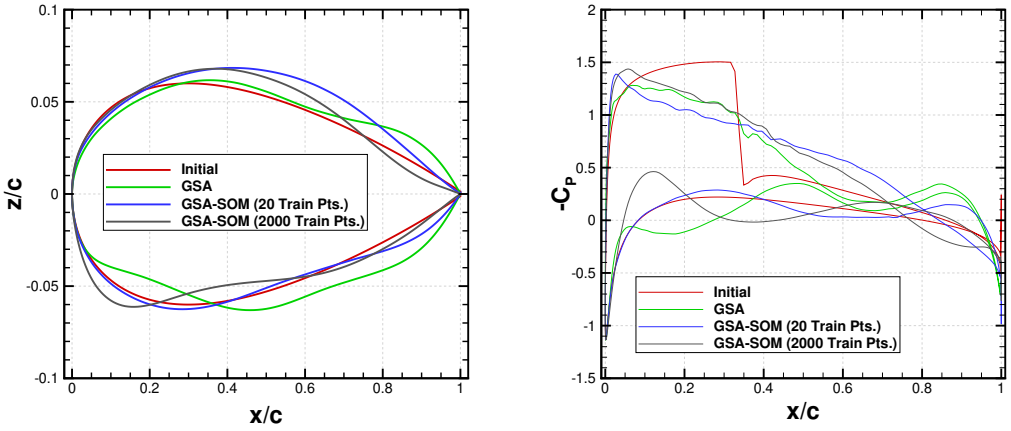


Fig. 11 Comparison of optimum shapes and pressure distributions for inviscid case 1.

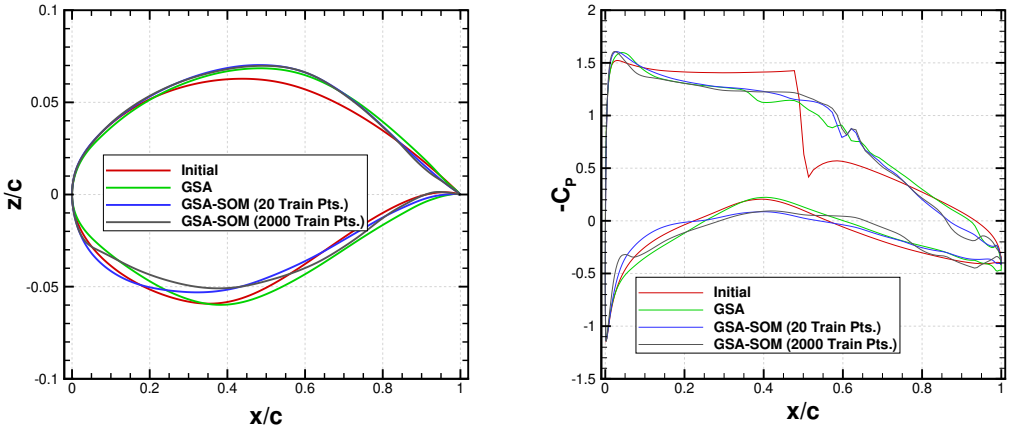


Fig. 12 Comparison of optimum shapes and pressure distributions for inviscid case 2.

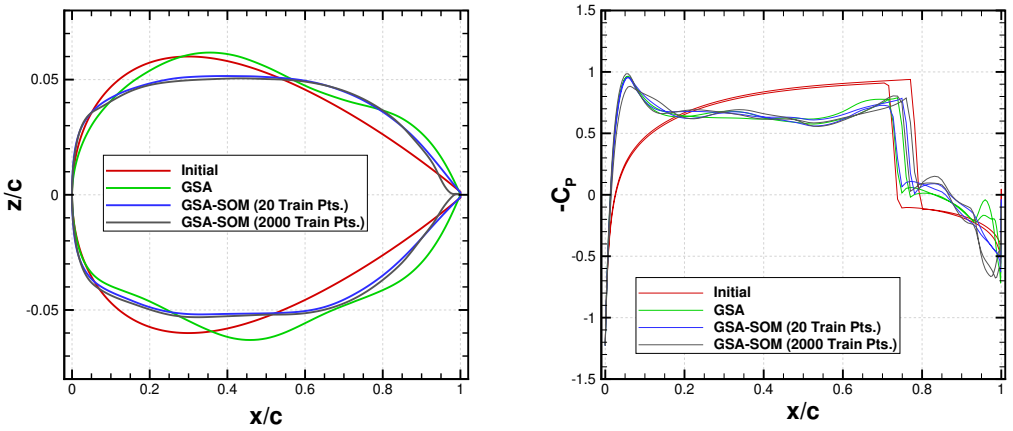


Fig. 13 Comparison of optimum shapes and pressure distributions for inviscid case 3.

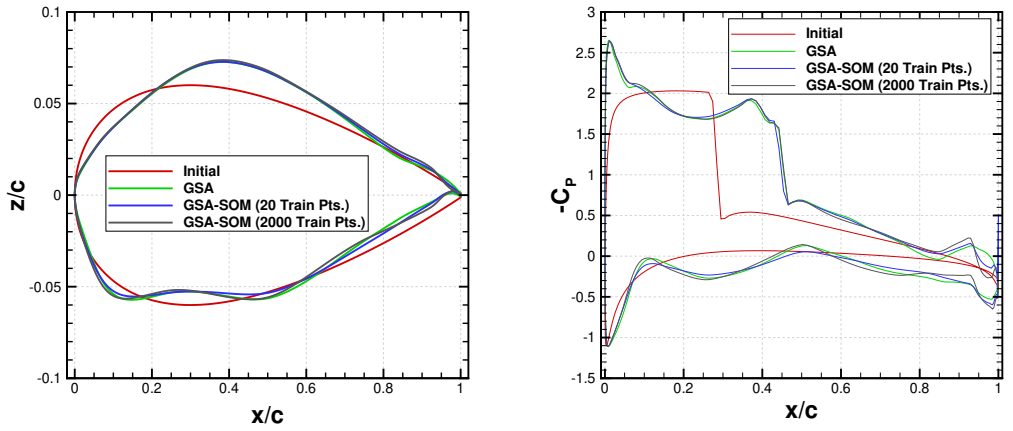


Fig. 14 Comparison of optimum shapes and pressure distributions for inviscid case 4.

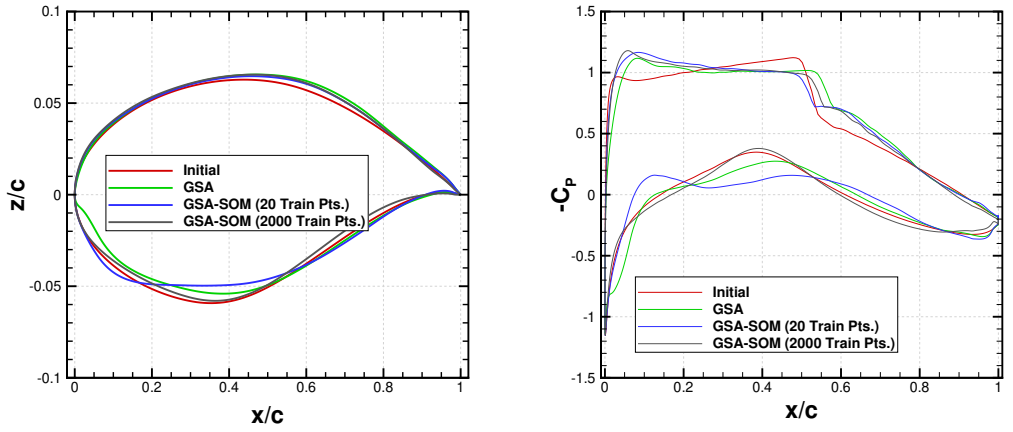


Fig. 15 Comparison of optimum shapes and pressure distributions for viscous case.

VI. Conclusions

The potential of machine learning for guiding a global search algorithm in optimisation problems has been considered and presented here, by coupling a Self-Organising Map (SOM) to a Gravitational Search Algorithm (GSA). Two different training strategies have been analysed and their efficiency compared first against a set of benchmark analytical functions typically used in optimisation and finally in a set of two-dimensional aerodynamic shape optimisation problems.

The Self-Organising Map-guided Gravitational Search Algorithm (GSA-SOM) outperformed the GSA and Particle Swarm Optimisation algorithms in most of the unconstrained benchmark functions analysed. GSA-SOM was also shown to outperform GSA for constrained aerodynamic optimisation problems, once again proving its robustness and superior performance.

Usage of a large number of data points to train the SOM was observed to result in a reduction of the modality of the search space through mapping on the SOM neural network, whereas a smaller number of training points proved to accelerate convergence in optimisation problems where modality of the search space is not as exacerbated.

The GSA-SOM algorithm, although resulting in a moderate increase in computational cost, due to the training costs of the SOM structure, was shown to lend itself well to parallelisation, and exhibited the same time complexity as the classical GSA algorithm, confirming it as a robust alternative for high-cost optimisation problems.

References

[1] Trana, K., "Finite element simulation of the tube hydroforming process—bending, preforming and hydroforming," *Journal of Materials Processing Technology*, Vol. 127, No. 3, 2002, pp. 401 – 408. https://doi.org/10.1016/S0924-0136(02)00432-6.

[2] Mital, M., and Pidaparti, R. M., "Breast Tumor Simulation and Parameters Estimation Using Evolutionary Algorithms," *Modelling and Simulation in Engineering*, , No. 1687-5591, 2008. https://doi.org/10.1155/2008/756436.

[3] Price, D. J., Wurster, J., Tricco, T. S., Nixon, C., Toupin, S., Pettitt, A., Chan, C., Mentiplay, D., Laibe, G., Glover, S., and et al., "Phantom: A Smoothed Particle Hydrodynamics and Magnetohydrodynamics Code for Astrophysics," *Publications of the Astronomical Society of Australia*, Vol. 35, 2018. https://doi.org/10.1017/pasa.2018.25.

[4] (Henry) Jia, Z., Lee, S., Sharma, K., and Brentner, K. S., "Aeroacoustic Analysis of a Lift-Offset Coaxial Rotor Using High-Fidelity CFD/CSD Loose Coupling Simulation," *Journal of the American Helicopter Society*, Vol. 65, No. 1, 2020, pp. 1–15. https://doi.org/10.4050/JAHS.65.012011.

[5] Vanderplaats, G. N., "Numerical Optimization Techniques," *Computer Aided Optimal Design: Structural and Mechanical Systems. NATO ASI Series (Series F: Computer and Systems Sciences)*, Vol. 27, edited by C. A. M. Soares, Springer, Berlin, Heidelberg, 1987, pp. 197–239. https://doi.org/10.1007/978-3-642-83051-8_5.

[6] Vanderplaats, G. N., *Multidiscipline design optimization*, Vanderplaats Research & Development, Inc., Monterey, CA, 2007.

[7] Gill, P. E., Murray, W., and Saunders, M. A., "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM Rev.*, Vol. 47, 2005, pp. 99–131.

[8] Tits, A., Lawrence, C., and L. Zhou, J., "User's Guide for CFSQP Version 2.5: A C Code for Solving (Large Scale) Constrained Nonlinear (Minimax) Optimization Problems, Generating Iterates Satisfying All Inequality Constraints," *Technical report TR-94-16r1, University of Maryland, College Park*, 1999.

[9] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P., "Optimization by Simulated Annealing," *Science*, Vol. 220, No. 4598, 1983, pp. 671–680. https://doi.org/10.1126/science.220.4598.671.

[10] Tayarani-N, M. H., and Akbarzadeh-T, M. R., "Magnetic Optimization Algorithms a new synthesis," *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, 2008, pp. 2659–2664. https://doi.org/10.1109/CEC.2008.4631155.

[11] Rashedi, E., Nezamabadi-pour, H., and Saryazdi, S., "GSA: A Gravitational Search Algorithm," *Information Sciences*, Vol. 179, No. 13, 2009, pp. 2232 – 2248. https://doi.org/10.1016/j.ins.2009.03.004.

[12] Zhou, Y., Luo, Q., and Chen, H., "A Novel Differential Evolution Invasive Weed Optimization Algorithm for Solving Nonlinear Equations Systems," *Journal of Applied Mathematics*, Vol. 2013, No. 1110-757X, 2013. https://doi.org/10.1155/2013/757391.

[13] Kennedy, J., and Eberhart, R., "Particle swarm optimization," *Proceedings of ICNN'95 - International Conference on Neural Networks*, Vol. 4, 1995, pp. 1942–1948. https://doi.org/10.1109/ICNN.1995.488968.

[14] Yang, X.-S., "Firefly Algorithms for Multimodal Optimization," *Stochastic Algorithms: Foundations and Applications*, edited by O. Watanabe and T. Zeugmann, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 169–178. https://doi.org/10.1007/978-3-642-04944-6_14.

[15] Duan, H., and Qiao, P., "Pigeon-inspired optimization: A new swarm intelligence optimizer for air robot path planning," *International Journal of Intelligent Computing and Cybernetics*, Vol. 7, 2014, pp. 24–37. https://doi.org/10.1108/IJICC-02-2014-0005.

[16] Forrester, A. I., Sóbester, A., and Keane, A. J., "Multi-fidelity optimization via surrogate modelling," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, Vol. 463, No. 2088, 2007, pp. 3251–3269. https://doi.org/10.1098/rspa.2007.1900.

[17] Ong, Y. S., Nair, P. B., and Keane, A. J., "Evolutionary Optimization of Computationally Expensive Problems via Surrogate Modeling," *AIAA Journal*, Vol. 41, No. 4, 2003, pp. 687–696. https://doi.org/10.2514/2.1999.

[18] Audet, C., Denni, J., Moore, D., Booker, A., and Frank, P., "A surrogate-model-based method for constrained optimization," *8th Symposium on Multidisciplinary Analysis and Optimization*, American Institute of Aeronautics and Astronautics, 2000. https://doi.org/10.2514/6.2000-4891.

[19] Bevan, R. L. T., Poole, D. J., Allen, C. B., and Rendall, T. C. S., "Adaptive Surrogate-Based Optimization of Vortex Generators for Tiltrotor Geometry," *Journal of Aircraft*, Vol. 54, No. 3, 2017, pp. 1011–1024. https://doi.org/10.2514/1.C033838.

[20] Bhosekar, A., and Ierapetritou, M., "Advances in surrogate based modeling, feasibility analysis, and optimization: A review," *Computers & Chemical Engineering*, Vol. 108, 2018, pp. 250 – 267. https://doi.org/10.1016/j.compchemeng.2017.09.017.

[21] Leskovec, J., Rajaraman, A., and Ullman, J. D., *Dimensionality Reduction*, 2nd ed., Cambridge University Press, 2014, pp. 384–414. https://doi.org/10.1017/CBO9781139924801.012.

[22] LeGresley, P., and Alonso, J., "Airfoil design optimization using reduced order models based on proper orthogonal decomposition," *Fluids 2000 Conference and Exhibit*, American Institute of Aeronautics and Astronautics, 2000. https://doi.org/10.2514/6.2000-2545.

[23] LeGresley, P., and Alonso, J., "Improving the Performance of Design Decomposition Methods with POD," *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2004. https://doi.org/10.2514/6.2004-4465.

[24] Oyama, A., Nonomura, T., and Fujii, K., "Data Mining of Pareto-Optimal Transonic Airfoil Shapes Using Proper Orthogonal Decomposition," *Journal of Aircraft*, Vol. 47, No. 5, 2010, pp. 1756–1762. https://doi.org/10.2514/1.C000264.

[25] Alexander, M. J., Allison, J. T., and Papalambros, P. Y., "Reduced representations of vector-valued coupling variables in decomposition-based design optimization," *Structural and Multidisciplinary Optimization*, Vol. 44, No. 3, 2011, pp. 379–391. https://doi.org/10.1007/s00158-011-0636-9.

[26] Poole, D. J., Allen, C. B., and Rendall, T. C. S., "Metric-Based Mathematical Derivation of Efficient Airfoil Design Variables," *AIAA Journal*, Vol. 53, No. 5, 2015, pp. 1349–1361. https://doi.org/10.2514/1.J053427.

[27] Eshragh, F., Pooyandeh, M., and Marceau, D. J., "Automated negotiation in environmental resource management: Review and assessment," *Journal of Environmental Management*, Vol. 162, 2015, pp. 148 – 157. https://doi.org/10.1016/j.jenvman.2015.07.051.

[28] Kohonen, T., "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, Vol. 43, No. 1, 1982, pp. 59–69. https://doi.org/10.1007/BF00337288.

[29] Kohonen, T., "Essentials of the self-organizing map," *Neural Networks*, Vol. 37, 2013, pp. 52 – 65. https://doi.org/10.1016/j.neunet.2012.09.018.

[30] Kohonen, T., *Data Management by Self-Organizing Maps*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 309–332. https://doi.org/10.1007/978-3-540-68860-0_15.

[31] Yin, H., *Learning Nonlinear Principal Manifolds by Self-Organising Maps*, Springer, 2007, Vol. 60, pp. 68–95. https://doi.org/10.1007/978-3-540-73750-6_3.

[32] Smith, A. J., "Applications of the self-organising map to reinforcement learning," *Neural Networks*, Vol. 15, No. 8, 2002, pp. 1107 – 1124. https://doi.org/10.1016/S0893-6080(02)00083-7.

[33] Malone, J., McGarry, K., Wermter, S., and Bowerman, C., "Data Mining Using Rule Extraction from Kohonen Self-Organising Maps," *Neural Comput. Appl.*, Vol. 15, No. 1, 2006, pp. 9–17. https://doi.org/10.1007/s00521-005-0002-1.

[34] Chen, J., Chen, J., and Zhang, X., "Reliability Prediction Model of Aircraft using Self-Organizing Map," *2007 IEEE International Conference on Automation and Logistics*, 2007, pp. 680–683. https://doi.org/10.1109/ICAL.2007.4338650.

[35] Jain, A. K., Murty, M. N., and Flynn, P. J., "Data Clustering: A Review," *ACM Comput. Surv.*, Vol. 31, No. 3, 1999, pp. 264–323. https://doi.org/10.1145/331499.331504.

[36] Raptodimos, Y., and Lazakis, I., "Using artificial neural network-self-organising map for data clustering of marine engine condition monitoring applications," *Ships and Offshore Structures*, Vol. 13, No. 6, 2018, pp. 649–656. https://doi.org/10.1080/17445302.2018.1443694.

[37] Ilc, N., and Dobnikar, A., "Gravitational Clustering of the Self-Organizing Map," *Adaptive and Natural Computing Algorithms*, edited by A. Dobnikar, U. Lotrič, and B. Šter, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 11–20. https://doi.org/10.1007/978-3-642-20267-4_2.

[38] Gonsalves, T., and Nishimoto, Y., "Data Clustering through Particle Swarm Optimization Driven Self-Organizing Maps," *Intelligence in the Era of Big Data*, edited by R. Intan, C.-H. Chi, H. N. Palit, and L. W. Santoso, Springer Berlin Heidelberg, Berlin, Heidelberg, 2015, pp. 212–219. https://doi.org/10.1007/978-3-662-46742-8_20.

[39] Mu-Chun Su, Yu-Xiang Zhao, and Lee, J., "SOM-based optimization," *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, Vol. 1, 2004, pp. 781–786. https://doi.org/10.1109/IJCNN.2004.1380019.

[40] Lawrence, R. D., Almasi, G. S., and Rushmeier, H. E., "A Scalable Parallel Algorithm for Self-Organizing Maps with Applications to Sparse Data Mining Problems," *Data Mining and Knowledge Discovery*, Vol. 3, No. 2, 1999, pp. 171–195. https://doi.org/10.1023/A:1009817804059.

[41] Takatsuka, M., and Bui, M., "Parallel Batch Training of the Self-Organizing Map Using OpenCL," *Neural Information Processing. Models and Applications*, edited by K. W. Wong, B. S. U. Mendis, and A. Bouzerdoum, Springer, Berlin, Heidelberg, 2010, pp. 470–476. https://doi.org/10.1007/978-3-642-17534-3_58.

[42] Rashedi, E., Nezamabadi-pour, H., and Saryazdi, S., "Filter modeling using gravitational search algorithm," *Engineering Applications of Artificial Intelligence*, Vol. 24, No. 1, 2011, pp. 117 – 122. https://doi.org/10.1016/j.engappai.2010.05.007.

[43] Sun, S., and Peng, Q., "A hybrid PSO-GSA strategy for high-dimensional optimization and microarray data clustering," *2014 IEEE International Conference on Information and Automation (ICIA)*, 2014, pp. 41–46. https://doi.org/10.1109/ICInfA.2014.6932623.

[44] Shi, Y., and Eberhart, R., "A modified particle swarm optimizer," *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence*, 1998, pp. 69–73. https://doi.org/10.1109/ICEC.1998.699146.

[45] Masters, D. A., Taylor, N. J., Rendall, T. C. S., Allen, C. B., and Poole, D. J., "Geometric Comparison of Aerofoil Shape Parameterization Methods," *AIAA Journal*, Vol. 55, No. 5, 2017, pp. 1575–1589. https://doi.org/10.2514/1.J054943.

[46] Chernukhin, O., and Zingg, D. W., "Multimodality and Global Optimization in Aerodynamic Design," *AIAA Journal*, Vol. 51, No. 6, 2013, pp. 1342–1354. https://doi.org/10.2514/1.J051835.

[47] Slooff, J. W., Schmidt, W., Papailiou, K. D., Bucciantini, G., Van der Vooren, J., Ashill, P. R., Holst, T. L., and Boppe, C. W., "Technical Status Review on Drag Prediction and Analysis from Computational Fluid Dynamics: State of the Art," AGARD Advisory Report 256, Advisory Group for Aerospace Research and Development, 1989.

[48] Golub, G., and Kahan, W., "Calculating the Singular Values and Pseudo-Inverse of a Matrix," *Journal of the Society for Industrial and Applied Mathematics Series B Numerical Analysis*, Vol. 2, No. 2, 1965, pp. 205–224. https://doi.org/10.1137/0702016.

[49] Rendall, T. C. S., and Allen, C. B., "Unified fluid–structure interpolation and mesh motion using radial basis functions," *International Journal for Numerical Methods in Engineering*, Vol. 74, No. 10, 2008, pp. 1519–1559. https://doi.org/10.1002/nme.2219.

[50] Wendland, H., *Scattered Data Approximation*, Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, 2004. https://doi.org/10.1017/CBO9780511617539.

[51] Buhmann, M. D., *Radial Basis Functions: Theory and Implementations*, Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, 2003. https://doi.org/10.1017/CBO9780511543241.

[52] van Leer, B., "Flux-vector splitting for the Euler equations," *Eighth International Conference on Numerical Methods in Fluid Dynamics*, edited by E. Krause, Springer Berlin Heidelberg, Berlin, Heidelberg, 1982, pp. 507–512. https://doi.org/10.1007/3-540-11948-5_66.

[53] Spalart, P. R., and Allmaras, S., "A one-equation turbulence model for aerodynamic flows," *30th Aerospace Sciences Meeting and Exhibit*, 1992. https://doi.org/10.2514/6.1992-439.

[54] Allmaras, S., Johnson, F., and Spalart, P., "Modifications and clarifications for the implementation of the Spalart-Allmaras turbulence model," *Seventh International Conference on Computational Fluid Dynamics (ICCFD7)*, 2012, pp. 1–11.

[55] Allen, C. B., "Multigrid convergence of inviscid fixed- and rotary-wing flows," *International Journal for Numerical Methods in Fluids*, Vol. 39, No. 2, 2002, pp. 121–140. https://doi.org/10.1002/fld.282.

[56] Allen, C., "Chimera volume grid generation within the EROS code," *Proceedings of The Institution of Mechanical Engineers Part G-journal of Aerospace Engineering - PROC INST MECH ENG G-J A E*, Vol. 214, 2000, pp. 125–141. https://doi.org/10.1243/0954410001531962.

[57] Allen, C. B., "Towards Automatic Structured Multiblock Mesh Generation using Improved Transfinite Interpolation," *International Journal for Numerical Methods in Engineering*, Vol. 74, No. 5, 2008, pp. 697–733. https://doi.org/10.1002/nme.2170.

[58] Poole, D. J., Allen, C. B., and Rendall, T. C. S., "High-fidelity aerodynamic shape optimization using efficient orthogonal modal design variables with a constrained global optimizer," *Computers and Fluids*, Vol. 143, 2017, pp. 1 – 15. https://doi.org/10.1016/j.compfluid.2016.11.002.

[59] Poole, D. J., Allen, C. B., and Rendall, T., "A Constrained Global Optimization Framework," *14th AIAA Aviation Technology, Integration, and Operations Conference*, 2014. https://doi.org/10.2514/6.2014-2034.

[60] Poole, D. J., Allen, C. B., and Rendall, T. C. S., "A generic framework for handling constraints with agent-based optimization algorithms and application to aerodynamic design," *Optimization and Engineering*, Vol. 18, No. 3, 2017, pp. 659–691. https://doi.org/10.1007/s11081-016-9343-0.